

OSELAS.BSP()

Phytec phyCORE-PXA270 RTG

PHYTEC



Quickstart Manual

<http://www.oselas.com>

© 2009 by Pengutronix, Hildesheim

1143 2009-02-02 10:53:45 +0100 (Mon, 02 Feb 2009)

Contents

I	OSELAS Quickstart for Phytec phyCORE-PXA270	4
1	Getting a working Environment	5
1.1	Download Software Components	5
1.2	PTXdist Installation	5
1.2.1	Main parts of PTXdist	5
1.2.2	Extracting the Sources	6
1.2.3	Prerequisites	7
1.2.4	Configuring PTXdist	8
1.3	Toolchains	9
1.3.1	Using Existing Toolchains	10
1.3.2	Building a Toolchain	10
1.3.3	Freezing the Toolchain	12
2	Building phyCORE-PXA270's root filesystem	13
2.1	Extracting	13
2.2	Selecting a Software Platform	14
2.3	Selecting a Hardware Platform	14
2.4	Selecting a Toolchain	14
2.5	Building the Root Filesystem	15
3	Bootng Linux	16
3.1	Target Side Preparation	17
3.2	Stand-Alone Bootng Linux	18
3.2.1	Development Host Preparations	18
3.2.2	Preparations on the Embedded Board	18
3.2.3	Bootng the Embedded Board	19
3.3	Remote-Bootng Linux	19
3.3.1	Development Host Preparations	19
3.3.2	Preparations on the Embedded Board	20
3.3.3	Bootng the Embedded Board	20
4	Accessng Peripherals	21
4.1	NOR Flash	21
4.2	Kernel Modules	22
4.3	PWM Units	23
4.4	GPIO	24
4.5	GPIO Events	25

Contents

4.6	Network	25
4.7	LCD Graphics	25
4.8	LCD Backlight	25
4.9	SPI	26
4.10	GPIO Expander	26
4.11	AC97 Based Audio	26
4.11.1	Sound Output	26
4.11.2	Sound Record	27
4.11.3	Advanced Sound Handling	28
4.12	AC97 Based Touchscreen	28
4.13	USB Host Controller	30
4.14	USB Gadget	30
4.15	I ² C Master	32
4.15.1	I ² C Realtime Clock (RTC8564)	32
4.15.2	I ² C device 24W32	32
4.16	Matrix keypad support	33
4.17	Status LEDs	33
4.18	Camera Interface	33
4.19	Using the Camera Support	33
4.20	Turn on Camera Support In System	34
4.21	Accessing Camera With GStreamer	35
4.22	Basic Usage Of GStreamer	36
4.22.1	Simple Usage Example	36
4.22.2	Simple Monochrome Usage Example	37
4.22.3	Simple Colour Usage Example	37
4.23	Advanced Usage Of GStreamer	37
4.23.1	Manual Setting for Frame Rate and Framesize	38
4.23.2	Manipulate Input Frame Size with Plugins	38
4.23.3	Manipulate Picture's Orientation	39
4.23.4	Using other Sinks than the Framebuffer	39
5	Getting help	41
5.1	Mailing Lists	41
5.2	News Groups	41
5.3	Chat/IRC	42
5.4	Miscellaneous	42
5.5	phyCORE-PXA270 Support Maillist	42
5.6	Commercial Support	42

Part I

OSELAS Quickstart for Phytec phyCORE-PXA270

1 Getting a working Environment

1.1 Download Software Components

In order to follow this manual, some software archives are needed. There are several possibilities how to get these: either as part of an evaluation board package or by downloading them from the Pengutronix web site.

The central place for OSELAS related documentation is <http://www.oselas.com>. This website provides all required packages and documentation (at least for software components which are available to the public).

To build OSELAS.BSP-Phytec-phyCORE-g, the following archives have to be available on the development host:

- ptxdist-1.99.10.tgz
- ptxdist-1.99.10-patches.tgz
- OSELAS.BSP-Phytec-phyCORE-g.tar.gz
- OSELAS.Toolchain-1.99.2.tar.bz2

If they are not available on the development system yet, it is necessary to get them.

1.2 PTXdist Installation

The PTXdist build system can be used to create a root filesystem for embedded Linux devices. In order to start development with PTXdist it is necessary to install the software on the development system.

This chapter provides information about how to install and configure PTXdist on the development host.

1.2.1 Main parts of PTXdist

The most important software component which is necessary to build an OSELAS.BSP() board support package is the ptxdist tool. So before starting any work we'll have to install PTXdist on the development host.

PTXdist consists of the following parts:

The ptxdist Program: ptxdist is installed on the development host during the installation process. ptxdist is called to trigger any action, like building a software packet, cleaning up the tree etc. Usually the ptxdist program is used in a *workspace* directory, which contains all project relevant files.

A Configuration System: The config system is used to customize a *configuration*, which contains information about which packages have to be built and which options are selected.

Patches: Due to the fact that some upstream packages are not bug free – especially with regard to cross compilation – it is often necessary to patch the original software. PTXdist contains a mechanism to automatically apply patches to packages. The patches are bundled into a separate archive. Nevertheless, they are necessary to build a working system.

Package Descriptions: For each software component there is a "recipe" file, specifying which actions have to be done to prepare and compile the software. Additionally, packages contain their configuration snippet for the config system.

Toolchains: PTXdist does not come with a pre-built binary toolchain. Nevertheless, PTXdist itself is able to build toolchains, which are provided by the OSELAS.Toolchain() project. More in-deep information about the OSELAS.Toolchain() project can be found here: http://www.pengutronix.de/oselas/toolchain/index_en.html

Board Support Package This is an optional component, mostly shipped aside with a piece of hardware. There are various BSP available, some are generic, some are intended for a specific hardware.

1.2.2 Extracting the Sources

To install PTXdist, at least two archives have to be extracted:

ptxdist-1.99.10.tgz The PTXdist software itself.

ptxdist-1.99.10-patches.tgz All patches against upstream software packets (known as the 'patch repository').

ptxdist-1.99.10-projects.tgz Generic projects (optional), can be used as a starting point for self-built projects.

The PTXdist and patches packets have to be extracted into some temporary directory in order to be built before the installation, for example the `local/` directory in the user's home. If this directory does not exist, we have to create it and change into it:

```
~# cd
~# mkdir local
~# cd local
```

Next steps are to extract the archives:

```
~/local# tar -zxf ptxdist-1.99.10.tgz
~/local# tar -zxf ptxdist-1.99.10-patches.tgz
```

and if required the generic projects:

```
~/local# tar -zxf ptxdist-1.99.10-projects.tgz
```

If everything goes well, we now have a PTXdist-1.99.10 directory, so we can change into it:

```
~/local# cd ptxdist-1.99.10
~/local/ptxdist-1.99.10# ls -l
```

```
total 455
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 autoconf/
-rwxr-xr-x  1 jb users    28 29. Sep 17:32 autogen.sh*
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 bin/
-rw-r--r--  1 jb users 115470 29. Sep 17:32 ChangeLog
drwxr-xr-x  5 jb users  1024 29. Sep 17:32 config/
-rwxr-xr-x  1 jb users 222451 29. Sep 17:34 configure*
-rw-r--r--  1 jb users  11445 29. Sep 17:34 configure.ac
-rw-r--r--  1 jb users  18361 29. Sep 17:32 COPYING
-rw-r--r--  1 jb users   3499 29. Sep 17:32 CREDITS
drwxr-xr-x  2 jb users  1024 29. Sep 17:30 debian/
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 Documentation/
drwxr-xr-x  7 jb users  1024 29. Sep 17:32 generic/
-rw-r--r--  1 jb users    58 29. Sep 17:32 INSTALL
-rw-r--r--  1 jb users  2150 29. Sep 17:32 Makefile.in
drwxr-xr-x 159 jb users  4096 29. Sep 17:31 patches/
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 platforms/
drwxr-xr-x  4 jb users  1024 29. Sep 17:30 plugins/
-rw-r--r--  1 jb users  4091 29. Sep 17:32 README
-rw-r--r--  1 jb users   691 29. Sep 17:32 REVISION_POLICY
drwxr-xr-x  6 jb users 28672 29. Sep 17:32 rules/
drwxr-xr-x  6 jb users  1024 29. Sep 17:30 scripts/
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 tests/
-rw-r--r--  1 jb users 33418 29. Sep 17:32 TODO
```

1.2.3 Prerequisites

Before PTXdist can be installed it has to be checked if all necessary programs are installed on the development host. The configure script will stop if it discovers that something is missing.

The PTXdist installation is based on GNU autotools, so the first thing to be done now is to configure the packet:

```
~/local/ptxdist-1.99.10# ./configure
```

This will check your system for required components PTXdist relies on. If all required components are found the output ends with:

```
[...]
checking whether /usr/bin/patch will work... yes

configure: creating ./config.status
config.status: creating Makefile
config.status: creating scripts/ptxdist_version.sh
config.status: creating rules/ptxdist-version.in

ptxdist version 1.99.10 configured.
Using '/usr/local' for installation prefix.
```

Report bugs to ptxdist@pengutronix.de

Without further arguments PTXdist is configured to be installed into `/usr/local`, which is the standard location for user installed programs. To change the installation path to anything non-standard, we use the `--prefix` argument to the `configure` script. The `--help` option offers more information about what else can be changed for the installation process.

The installation paths are configured in a way that several PTXdist versions can be installed in parallel. So if an old version of PTXdist is already installed there is no need to remove it.

One of the most important tasks for the `configure` script is to find out if all the programs PTXdist depends on are already present on the development host. The script will stop with an error message in case something is missing. If this happens, the missing tools have to be installed from the distribution before re-running the `configure` script.

When the `configure` script is finished successfully, we can now run

```
~/local/ptxdist-1.99.10# make
```

All program parts are being compiled, and if there are no errors we can now install PTXdist into its final location. In order to write to `/usr/local`, this step has to be performed as user `root`:

```
~/local/ptxdist-1.99.10# sudo make install
[enter root password]
[...]
```

If we don't have root access to the machine it is also possible to install into some other directory with the `--prefix` option. We need to take care that the `bin/` directory below the new installation dir is added to our `$PATH` environment variable (for example by exporting it in `~/ .bashrc`).

The installation is now done, so the temporary folder may now be removed:

```
~/local/ptxdist-1.99.10# cd
~# rm -fr local
```

1.2.4 Configuring PTXdist

When using PTXdist for the first time, some setup properties have to be configured. Two settings are the most important ones: Where to store the source packages and if a proxy must be used to gain access to the world wide web.

Run PTXdist's setup:

```
~# ptxdist setup
```

Due to PTXdist is working with sources only, it needs various source archives from the world wide web. If these archives are not present on our host, PTXdist starts the `wget` command to download them on demand.

1.2.4.1 Proxy Setup

To do so, an internet access is required. If this access is managed by a proxy `wget` command must be advised to use it. PTXdist can be configured to advice the `wget` command automatically: Navigate to entry *Proxies* and enter the required addresses and ports to access the proxy in the form:

<protocol>://<address>:<port>

1.2.4.2 Source Archive Location

Whenever PTXdist downloads source archives it stores it project locally. If we are working with more than one project, every project would download its own required archives. To share all source archives between all projects PTXdist can be configured to use only one archive directory for all projects it handles: Navigate to menu entry *Source Directory* and enter the path to the directory where PTXdist should store archives to share between projects.

1.2.4.3 Generic Project Location

If we already installed the generic projects we should also configure PTXdist to know this location. If we already did so, we can use the command `ptxdist projects` to get a list of available projects and `ptxdist clone` to get a local working copy of a shared generic project.

Navigate to menu entry *Project Searchpath* and enter the path to projects that can be used in such a way. Here we can configure more than one path, each part can be delimited by a colon. For example for PTXdist's generic projects and our own previous projects like this:

```
/usr/local/lib/ptxdist-1.99.10/projects:/office/my_projects/ptxdist
```

Leave the menu and store the configuration. PTXdist is now ready for use.

1.3 Toolchains

Before we can start building our first userland we need a cross toolchain. On Linux, toolchains are no monolithic beasts. Most parts of what we need to cross compile code for the embedded target comes from the *GNU Compiler Collection*, `gcc`. The `gcc` packet includes the compiler frontend, `gcc`, plus several backend tools (`cc1`, `g++`, `ld` etc.) which actually perform the different stages of the compile process. `gcc` does not contain the assembler, so we also need the *GNU Binutils package* which provides lowlevel stuff.

Cross compilers and tools are usually named like the corresponding host tool, but with a prefix – the *GNU target*. For example, the cross compilers for ARM and powerpc may look like

- `arm-softfloat-linux-gnu-gcc`
- `powerpc-unknown-linux-gnu-gcc`

With these compiler frontends we can convert e.g. a C program into binary code for specific machines. So for example if a C program is to be compiled natively, it works like this:

```
~# gcc test.c -o test
```

To build the same binary for the ARM architecture we have to use the cross compiler instead of the native one:

```
~# arm-softfloat-linux-gnu-gcc test.c -o test
```

Also part of what we consider to be the "toolchain" is the runtime library (libc, dynamic linker). All programs running on the embedded system are linked against the libc, which also offers the interface from user space functions to the kernel.

The compiler and libc are very tightly coupled components: the second stage compiler, which is used to build normal user space code, is being built against the libc itself. For example, if the target does not contain a hardware floating point unit, but the toolchain generates floating point code, it will fail. This is also the case when the toolchain builds code for i686 CPUs, whereas the target is i586.

So in order to make things working consistently it is necessary that the runtime libc is identical with the libc the compiler was built against.

PTXdist doesn't contain a pre-built binary toolchain. Remember that it's not a distribution but a development tool. But it can be used to build a toolchain for our target. Building the toolchain usually has only to be done once. It may be a good idea to do that over night, because it may take several hours, depending on the target architecture and development host power.

1.3.1 Using Existing Toolchains

If a toolchain is already installed which is known to be working, the toolchain building step with PTXdist may be omitted.



The OSELAS.BoardSupport() Packages shipped for PTXdist have been tested with the OSELAS.Toolchains() built with the same PTXdist version. So if an external toolchain is being used which isn't known to be stable, a target may fail. Note that not all compiler versions and combinations work properly in a cross environment.

Every OSELAS.BoardSupport() Package checks for its OSELAS.Toolchain it's tested against, so using a different toolchain vendor requires an additional step:

Open the OSELAS.BoardSupport() Package menu with:

```
~# ptxdist platformconfig
```

and navigate to architecture --> toolchain and check for specific toolchain vendor. Clear this entry to disable the toolchain vendor check.

1.3.2 Building a Toolchain

PTXdist handles toolchain building as a simple project, like all other projects, too. So we can download the OSELAS.Toolchain bundle and build the required toolchain for the OSELAS.BoardSupport() Package.

A PTXdist project generally allows to build into some project defined directory; all OSELAS.Toolchain projects that come with PTXdist are configured to use the standard installation paths mentioned below.

All OSELAS.Toolchain projects install their result into /opt/OSELAS.Toolchain-1.99.2/.



Usually the `/opt` directory is not world writeable. So in order to build our OSELAS.Toolchain into that directory we need to use a root account to change the permissions. PTXdist detects this case and asks if we want to run `sudo` to do the job for us. Alternatively we can enter:

```
mkdir /opt/OSELAS.Toolchain-1.99.2
chown <username> /opt/OSELAS.Toolchain-1.99.2
chmod a+rw /opt/OSELAS.Toolchain-1.99.2.
```

We recommend to keep this installation path as PTXdist expects the toolchains at `/opt`. Whenever we go to select a platform in a project, PTXdist tries to find the right toolchain from data read from the platform configuration settings and a toolchain at `/opt` that matches to these settings. But that's for our convenience only. If we decide to install the toolchains at a different location, we still can use the *toolchain* parameter to define the toolchain to be used on a per project base.

1.3.2.1 Building the OSELAS.Toolchain for OSELAS.BSP-Phytec-phyCORE-9

To compile and install an OSELAS.Toolchain we have to extract the OSELAS.Toolchain archive, change into the new folder, configure the compiler in question and start the build.

The required compiler to build the OSELAS.BSP-Phytec-phyCORE-9 board support package is

```
arm-iwmmx-linux-gnueabi_gcc-4.3.2_glibc-2.8_binutils-2.18_kernel-2.6.27-sanitized
```

So the steps to build this toolchain are:

```
~# tar xf OSELAS.Toolchain-1.99.2.tar.bz2
~# cd OSELAS.Toolchain-1.99.2
~/OSELAS.Toolchain-1.99.2# ptxdist select ptxconfigs/\ 
> arm-iwmmx-linux-gnueabi_gcc-4.3.2_glibc-2.8_binutils-2.18_kernel-2.6.27-sanitized.ptxconfig
~/OSELAS.Toolchain-1.99.2# ptxdist go
```

At this stage we have to go to our boss and tell him that it's probably time to go home for the day. Even on reasonably fast machines the time to build an OSELAS.Toolchain is something like around 30 minutes up to a few hours.

Measured times on different machines:

- Single Pentium 2.5 GHz, 2 GiB RAM: about 2 hours
- Turion ML-34, 2 GiB RAM: about 1 hour 30 minutes
- Dual Athlon 2.1 GHz, 2 GiB RAM: about 1 hour 20 minutes
- Dual Quad-Core-Pentium 1.8 GHz, 8 GiB RAM: about 25 minutes

Another possibility is to read the next chapters of this manual, to find out how to start a new project.

When the OSELAS.Toolchain project build is finished, PTXdist is ready for prime time and we can continue with our first project.

1.3.3 Freezing the Toolchain

As we build and install this toolchain with regular user permissions we should modify the permissions as a last step to avoid any later manipulation. To do so we could set all toolchain files to read only or change recursively the owner of the whole installation to user root.

This is an important step for reliability. Do not omit it!

1.3.3.1 Building additional Toolchains

The OSELAS.Toolchain-1.99.2 bundle comes with various predefined toolchains. Refer the `ptxconfigs/` folder for other definitions. To build additional toolchains we only have to clean our current toolchain projekt, removing the current `selected_ptxconfig` link and creating a new one.

```
~/OSELAS.Toolchain-1.99.2# ptxdist clean
~/OSELAS.Toolchain-1.99.2# rm selected_ptxconfig
~/OSELAS.Toolchain-1.99.2# ptxdist select \ 
> ptxconfigs/any_other_toolchain_def.ptxconfig
~/OSELAS.Toolchain-1.99.2# ptxdist go
```

All toolchains will be installed side by side architecture dependend into directory

`/opt/OSELAS.Toolchain-1.99.2/architecture_part.`

Different toolchains for the same architecture will be installed side by side version dependend into directory

`/opt/OSELAS.Toolchain-1.99.2/architecture_part/version_part.`

2 Building phyCORE-PXA270's root filesystem

2.1 Extracting

In order to work with a PTXdist based project we have to extract the archive first.

```
~# tar -zxf OSELAS.BSP-Phytec-phyCORE-9.tar.gz
~# cd OSELAS.BSP-Phytec-phyCORE-9
```

PTXdist is project centric, so now after changing into the new directory we have access to all valid components.

```
~/OSELAS.BSP-Phytec-phyCORE-9# ls -l

total 48
-rw-r--r--  1 jbe ptx 5149 Feb  3 12:44 ChangeLog
-rw-r--r--  1 jbe ptx 1313 Feb  3 12:45 Kconfig
drwxr-xr-x 10 jbe ptx 4096 Feb  3 12:57 configs
drwxr-xr-x  3 jbe ptx 4096 Feb  3 12:45 documentation
drwxr-xr-x  4 jbe ptx 4096 Feb  3 13:00 local_src
drwxr-xr-x  9 jbe ptx 4096 Feb  3 12:44 patches
drwxr-xr-x  3 jbe ptx 4096 Feb  3 12:44 platforms
drwxr-xr-x  6 jbe ptx 4096 Feb  3 12:45 projectroot
drwxr-xr-x  3 jbe ptx 4096 Feb  3 12:56 protocols
drwxr-xr-x  4 jbe ptx 4096 Feb  3 12:44 rules
drwxr-xr-x  3 jbe ptx 4096 Feb  3 12:59 tests
```

Notes about some of the files and directories listed above:

ChangeLog Here you can read what has changed in this release. Note: This file does not always exist.

documentation If this BSP is one of our OSELAS BSPs, this directory contains the Quickstart you are currently reading in.

configs A multiplatform BSP contains configurations for more than one target. This directory contains the platform configuration files.

projectroot Contains files and configuration for the target's runtime. A running GNU/Linux system uses many text files for runtime configuration. Most of the time the generic files from the PTXdist installation will fit the needs. But if not, customized files are located in this directory.

rules If something special is required to build the BSP for the target it is intended for, then this directory contains these additional rules.

patches If some special patches are required to build the BSP for this target, then this directory contains these patches on a per package basis.

tests Contains test scripts for automated target setup.

2.2 Selecting a Software Platform

First of all we have to select a software platform for the userland configuration. This step defines what kind of applications will be built for the hardware platform. The OSELAS.BSP-Phytec-phyCORE-9 comes with a predefined configuration we select in the following step:

```
~/OSELAS.BSP-Phytec-phyCORE-9# ptxdist select \   
> configs/ptxconfig  
info: selected ptxconfig:  
      'configs/ptxconfig'
```

2.3 Selecting a Hardware Platform

Before we can build this BSP, we need to select one of the possible targets to build for. In this case we want to build for the phyCORE-PXA270:

```
~/OSELAS.BSP-Phytec-phyCORE-9# ptxdist platform \   
> configs/phyCORE-PXA270-1.99-10-1/platformconfig.pcm969  
info: selected platformconfig:  
      'configs/phyCORE-PXA270-1.99-10-1/platformconfig.pcm969'
```

Note: If you have installed the OSELAS.Toolchain() at its default location, PTXdist should already have detected the proper toolchain while selecting the platform. In this case it will output:

```
found and using toolchain:  
'/opt/OSELAS.Toolchain-1.99.2/arm-iwmmx-linux-gnueabi/  
  gcc-4.3.2-glibc-2.8-binutils-2.18-kernel-2.6.27-sanitized/bin'
```

If it fails you can continue to select the toolchain manually as mentioned in the next section. If this autodetection was successful, we can omit the steps of the section and continue to build the BSP.

2.4 Selecting a Toolchain

If not automatically detected, the last step in selecting various configurations is to select the toolchain to be used to build everything for the target.

```
~/OSELAS.BSP-Phytec-phyCORE-9# ptxdist toolchain \   
> /opt/OSELAS.Toolchain-1.99.2/arm-iwmmx-linux-gnueabi/\   
> gcc-4.3.2-glibc-2.8-binutils-2.18-kernel-2.6.27-sanitized/bin
```

2.5 Building the Root Filesystem

Now everything is prepared for PTXdist to compile the BSP. Starting the engines is simply done with:

```
~/OSELAS.BSP-Phytec-phyCORE-9# ptxdist go
```

PTXdist does now automatically find out from the `selected_ptxconfig` and `selected_platformconfig` files which packages belong to the project and starts compiling their *targetinstall* stages (that one that actually puts the compiled binaries into the root filesystem). While doing this, PTXdist finds out about all the dependencies between the packets and brings them into the correct order.

While the command `ptxdist go` is running we can watch it building all the different stages of a packet. In the end the final root filesystem for the target board can be found in the `platform-phyCORE-PXA270.PCM969/root/` directory and a bunch of **.ipk* packets in the `platform-phyCORE-PXA270.PCM969/packages/` directory, containing the single applications the root filesystem consists of.

3 Booting Linux

Now that there is a root filesystem in our workspace we'll have to make it visible to the phyCORE-PXA270. There are two possibilities to do this:

1. Booting from the development host, via network.
2. Making the root filesystem persistent in the onboard flash.

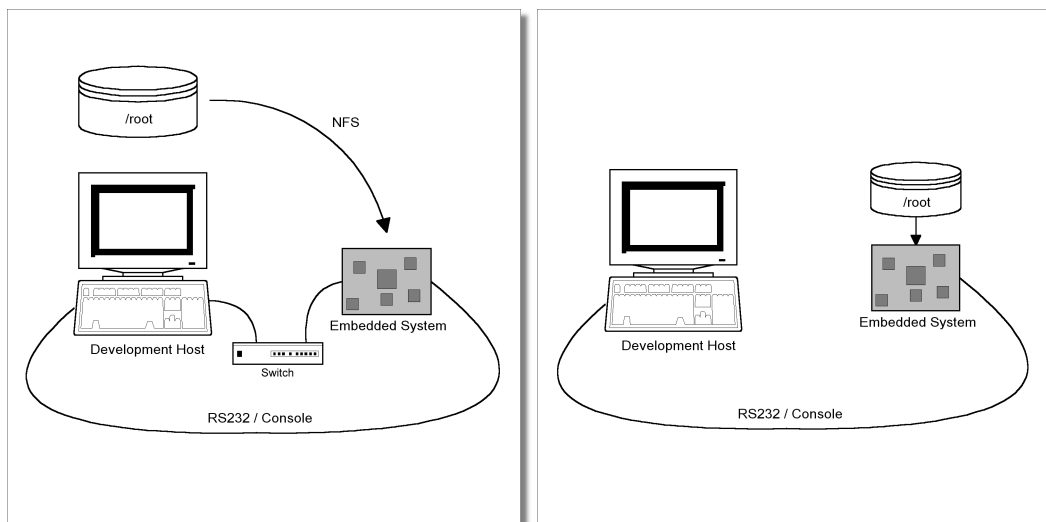


Figure 3.1: Booting the root filesystem, built with PTXdist, from the host via network and from flash.

Figure 3.1 shows both methods. On the left side the development host is connected to the phyCORE-PXA270 with a serial nullmodem cable and via ethernet; the embedded board boots into the bootloader, then issues a TFTP request on the network and boots the kernel from the TFTP server on the host. Then, after decompressing the kernel into the RAM and starting it, the kernel mounts it's root filesystem via NFS from the original location of the root/ directory in our PTXdist workspace.

The other way is to provide all needed components to run on the target itself. The Linux kernel and the root filesystem is persistent in target's flash. This means the only connection needed is the nullmodem cable to see what is happen on our target.

This chapter describes how to set up our target with features supported by PTXdist to simplify this challange.

3.1 Target Side Preparation

The phyCORE-PXA270 uses U-Boot as its bootloader. U-Boot can be customised with environment variables and scripts to support any boot constellation. OSELAS.BSP-Phytec-phyCORE-9 comes with a predefined environment setup to easily bring up the phyCORE-PXA270.

Usually the environment doesn't have to be set manually on our target. PTXdist comes with an automated setup procedure to achieve a correct environment on the target.

Due to the fact some of the values of these U-Boot's environment variables must meet our local network environment and development host settings we have to define them prior to running the automated setup procedure.

Note: At this point of time it makes sense to check if the serial connection is already working, because it is essential for any further step we will do.

We can try to connect to the target with our favorite terminal application (`minicom` or `kermit` for example). With a powered target we identify the correct physical serial port and ensure that the communication is working. Make sure to leave this terminal application to unlock the serial port prior to the next steps.

To set up development host and target specific value settings we run the command

```
~/OSELAS.BSP-Phytec-phyCORE-9# ptxdist boardsetup
```

We navigate to "Network Configuration" and replace the default settings with our local network settings. In the next step we also should check if the "Host's Serial Configuration" entries meet our local development host settings. Especially the "serial port" must correspond to our real physical connection.

"Exit" the dialouge and and save your new settings.

The command

```
~/OSELAS.BSP-Phytec-phyCORE-9# ptxdist test setenv
```

now will automatically set up a correct default environment on the phyCORE-PXA270.

It should output a line like this when it was successful:

```
=====
Please power on your board now!
=====
```

```
Logging into U-Boot.....OK
Setting new environment.....OK
Test finished successfully.
```

Note: If it fails, reading `platform-phyCORE-PXA270.PCM969/test.log` will give further information about why it has failed.

We now must restart the phyCORE-PXA270 to activate the new environment settings. Then we should run the `ping` command on the target's ip address to check if the network settings are working correctly on the target.

3.2 Stand-Alone Booting Linux

Usually, after working with the NFS-Root system for some time, the rootfs has to be made persistent in the on-board flash of the phyCORE-PXA270, without requiring the network infrastructure any more. The following sections describe the steps necessary to bring the rootfs into the onboard flash.

Only for preparation we need a network connection to the embedded board and a network aware bootloader which can fetch any data from a TFTP server.

After preparation is done, the phyCORE-PXA270 can work independently from the development host. We can "cut" the network (and serial cable) and the phyCORE-PXA270 will continue to work.

3.2.1 Development Host Preparations

If we already booted the phyCORE-PXA270 remotely (as described in the previous section), all of the development host preparations are done.

If not, then a TFTP server has to be installed and configured on the development host. The exact method of doing this is distribution specific; as the TFTP server is usually started by one of the inetd servers, the manual sections describing `inetd` or `xinetd` should be consulted.

Usually TFTP servers are using the `/tftpboot` directory to fetch files from, so if we want to push data files to this directory we have to make sure we are able to write there. As the access permissions are normally configured in a way to let only user `root` write to `/tftpboot`, we have to gain access.

3.2.2 Preparations on the Embedded Board

To boot phyCORE-PXA270 stand-alone, anything needed to run a Linux system must be locally accessible. So at this point of time we must replace any current content in phyCORE-PXA270's flash memory. To simplify this, OSELAS.BSP-Phytec-phyCORE-9 comes with an automated setup procedure for this step.

To use this procedure run the command

```
~# ptxdist test flash
```

Note: This command requires a serial and a network connection. The network connection can be cut after this step.

This command will automatically write a root filesystem to the correct flash partition on the phyCORE-PXA270. It only works if we previously have set up the environment variables successfully (described at page 17).

The command should output a line like this when it was successful:

```
=====
Please power on your board now!
=====
```

```
Logging into U-Boot.....OK
Flashing kernel.....OK
Flashing rootfs.....OK
Flashing oftree.....OK
Test finished successfully.
```

Note: If it fails, reading `platform-phyCORE-PXA270.PCM969/test.log` will give further information about why it has failed.

3.2.3 Booting the Embedded Board

To check that everything went successfully up to here, we can run the *boot* test.

```
~# ptxdist test boot

=====
Please power on your board now!
=====

Checking for U-Boot.....OK
Checking for Kernel.....OK
Checking for init.....OK
Checking for login.....OK
Test finished successfully.
```

This will check if the environment settings and flash partitioning are working as expected, so the target comes up in stand-alone mode up to the login prompt.

Note: If it fails, reading `platform-phyCORE-PXA270.PCM969/test.log` will give further information about why it has failed.

After the next reset or powercycle of the board, it should boot the kernel from the flash, start it and mount the root filesystem also from flash.

Note: The default login account is `root` with an empty password.

3.3 Remote-Booting Linux

The next method we want to try after building a root filesystem is the network-remote boot variant. All we need is a network interface on the embedded board and a network aware bootloader which can fetch the kernel from a TFTP server.

The network boot method has the advantage that we don't have to do any flashing at all to "see" a file on the target board: All we have to do is to copy it to some location in the `root/` directory and it simply "appears" on the embedded device. This is especially helpful during the development phase of a project, where things are changing frequently.

3.3.1 Development Host Preparations

On the development host a TFTP server has to be installed and configured. The exact method to do this is distribution specific; as the TFTP server is usually started by one of the `inetd` servers, the manual sections describing `inetd` or `xinetd` should be consulted.

Usually TFTP servers are using the `/tftpboot` directory to fetch files from, so if we want to push kernel images to this directory we have to make sure we are able to write there. As the access permissions are normally configured in a way to let only user `root` write to `/tftpboot` we have to gain access; a safe method is to use the `sudo(8)` command to push our kernel:

```
~# sudo cp platform-phyCORE-PXA270/images/linuximage /tftpboot/uImage-pcm027
```

The NFS server is not restricted to a certain filesystem location, so all we have to do on most distributions is to configure `/etc/exports` and export our root filesystem to the embedded network. In this example file the whole work directory is exported, and the "lab network" between the development host is 192.168.23.0, so the IP addresses have to be adapted to the local needs:

```
/home/<user>/work 192.168.23.0/255.255.255.0(rw,no_root_squash,sync)
```

Note: Replace `<user>` with your home directory name.

3.3.2 Preparations on the Embedded Board

We already provided the phyCORE-PXA270 with the default environment at page 17. So there is no additional preparation required here.

3.3.3 Booting the Embedded Board

The default environment settings coming with the OSELAS.BSP-Phytec-phyCORE-g has the possibility to boot from the internal flash or from the network. The definition what should happen after power on is made with the environment variable `boot_cmd`. The default setting is `run bcmd_flash` and will boot from flash.

To change this behavior we have to change the value of the `boot_cmd` environment variable to `run bcmd_net`.

```
uboot> setenv boot_cmd 'run bcmd_net'
uboot> saveenv
```

The next time the target will start it will use the network based booting mechanism.

4 Accessing Peripherals

The following sections provide an overview of the supported hardware components and their corresponding operating system drivers. Further changes can be ported on demand of the customer.

Phytec's phyCORE-PXA270 starter kit consists of the following individual boards:

1. The phyCORE-PXA270 module itself (PCM-027), containing the PXA270, RAM, flash, the GPIO expander chip and several other peripherals.
2. The starter kit baseboard (PCM969).
3. Camera kit.

To achieve maximum software re-use, the Linux kernel offers a sophisticated infrastructure, layering software components into board specific parts. The OSELAS.BSP() tries to modularize the kit features as far as possible; that means that when a customized baseboards or even customer specific module is developed, most of the software support can be re-used without error prone copy-and-paste. So the kernel code corresponding to the boards above can be found in

1. `arch/arm/mach-pxa/pcm027.c` for the CPU module
2. `arch/arm/mach-pxa/pcm969_990-baseboard.c` for the baseboard. With this file two common baseboards PCM969 and PCM990 are supported. The kernel commandline parameter can be used to select the board type during boot period. `board=pcm969` turns on board specific support for PCM969. The correct kernel command line should be set up in the u-boot environment. So mostly you don't have to change this.

In fact, software re-use is one of the most important features of the Linux kernel and especially of the ARM port, which always had to fight with an insane number of possibilities of the System-on-Chip CPUs.



Note that the huge variety of possibilities offered by the phyCORE modules makes it difficult to have a completely generic implementation on the operating system side. Nevertheless, the OSELAS.BSP() can easily be adapted to customer specific variants. In case of interest, contact the Pengutronix support (support@pengutronix.de) and ask for a dedicated offer.

The following sections provide an overview of the supported hardware components and their operating system drivers.

4.1 NOR Flash

Linux offers the Memory Technology Devices Interface (MTD) to access low level flash chips, directly connected to a SoC CPU.

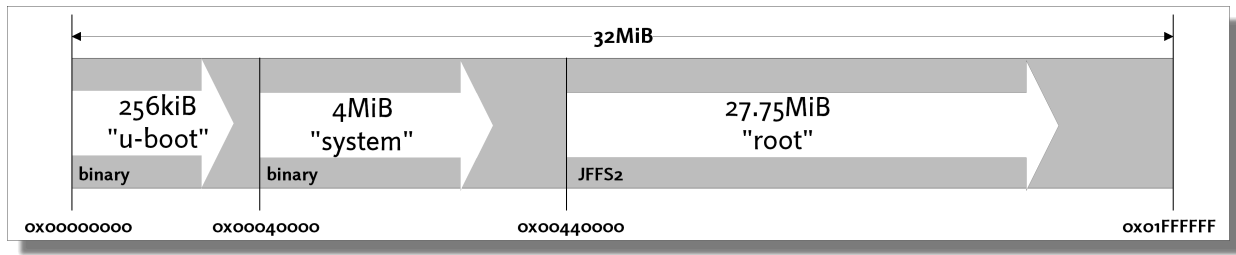


Figure 4.1: Current flash partitions

Modern kernels offer a method to define flash partitions on the kernel command line, using the "mtdparts" command line argument:

```
mtdparts=physmap-flash.0:256k(u-boot)ro,4096k(system),-(root)
```

This line, for example, specifies several partitions with their size and name which can be used as /dev/mtd0, /dev/mtd1 etc. from Linux. Additionally, this argument is also understood by reasonably new U-Boot bootloaders, so if there is any need to change the partitioning layout, the U-Boot environment is the only place where the layout has to be changed. In this section we assume that the standard configuration delivered with the OSELAS.BSP-Phytec-phyCORE-9 is being used.

From userspace the flash partitions can be accessed as

- /dev/mtdblock0 (U-Boot partition)
- /dev/mtdblock1 (Linux kernel)
- /dev/mtdblock2 (Linux rootfs partition)

Only /dev/mtdblock2 has a filesystem, so the other partitions cannot be mounted into the rootfs. The only way to access them is by pushing a prepared flash image into the corresponding /dev/mtd device node.

4.2 Kernel Modules

The OSELAS.BSP-Phytec-phyCORE-9 BSP uses a modularised kernel to support most of the hardware. This is the list of modules loaded at system startup as a default:

```
~# lsmod
Module                Size  Used by
uio_pdrv_genirq        3008   0
uio                    9080   1 uio_pdrv_genirq
leds_pwm              2496   0
leds_gpio             2624   0
pca953x               4160   0
pxa_camera            11872   0
videobuf_dma_sg       11332   1 pxa_camera
soc_camera            12676   1 pxa_camera
```

videobuf_core	16644	3	pxa_camera,videobuf_dma_sg,soc_camera
videodev	30944	1	soc_camera
at24	5952	0	
pxa2xx_spi	13312	0	
vfat	11296	0	
fat	51100	1	vfat
sd_mod	24244	0	
usb_storage	45664	0	
scsi_mod	101404	2	sd_mod,usb_storage
usbhid	17588	0	
hid	39460	1	usbhid
mousedev	13000	0	
ohci_hcd	19204	0	

4.3 PWM Units

The PXA270 has four PWM units which can be programmed individually. However, as the phyCORE-PXA270 has some hardware restrictions, not all of them can be used under all circumstances:

- PWM#0 is used to control the white PWM driven LED (for usage information see section 4.17)
- PWM#1 is used to control the green PWM driver LED (for usage information see section 4.17)
- PWM#2 is used for LCD Backlight brightness (see section 4.8)
- PWM#3 is not available

The PWM units can be controlled with sysfs entries, with which we can acquire a PWM unit, setup the period and output. Due to optimisation issue the four PWM Units are registered to two platform devices, each of which represents two PWM units. You can find the sysfs directories for the platform devices under `/sys/bus/platform/devices/pxa27x-pwm.0` and `/sys/bus/platform/devices/pxa27x-pwm.1`, while the first one represents PWM#0 and PWM#2, the second one PWM#1 and PWM#3. The entries we can use to control the PWM units are located under the corresponding platform device directories. For each PWM unit there are five entries:

- `.../acquire`
Before we can control a PWM unit, we must first acquire it to make sure that it is not used already by some one else. Just write the PWM unit number we want to acquire into this entry. Note that we can only acquire a PWM unit the platform device actually represents. e.g. you can acquire PWM#0 and PWM#2 with `/sys/bus/platform/devices/pxa27x-pwm.0?/acquire`. If we wish to control PWM#3, you'd have to use the acquire entry in `/sys/bus/platform/devices/pxa27x-pwm.1`. Once we have acquired a PWM unit, we can use the entries listed below to control it.
- `.../release`
Write a non-zero value into this entry to release our currently acquired PWM unit.
- `.../period`
This entry can be used to change the period of the PWM signal. The unit of the values being written here is Nanoseconds. Valid numbers are 100 ...10000000 (100 us ...10 ms). Default value is 78770.
- `.../maxlevel`
Use this entry to set the maximal level of our PWM output. Default value is 255.

- .../setlevel

Writing a value into this entry will set the pwm output level. The actual output of the PWM unit will be a percentage of the presetted maximal level.

Note: You can only control a PWM units directly only if it is not acquired by another driver already. At default the predefined PWM units listed above are all registered to a corresponding driver. On phyCORE-PXA270 PWM#0 and PWM#1 are used by the leds-pwm driver, while PWM#2 is acquired and locked by the backlight driver. If you wish to access this elements, please use the infrastructures provided by the individual drivers. For LCD backlight please consult section 4.8). For the PWM driven LEDs please see 4.17. If you wish to access the PWM unit manually, you have to unload the driver controlling the uni first before you acquire it through the user interface. We don't recommend this though.

4.4 GPIO

Like most modern System-on-Chip CPUs, the PXA270 has numerous GPIO pins, some of which can be used for general purpose operations. If the generic gpio driver is loaded it offers a special sysfs entry that can be used to map a pin for userspace usage.

The control files are all contained in /sys/class/gpio

```
~# ls /sys/class/gpio/
export      gpiochip129  gpiochip64    unexport
gpiochip0   gpiochip32      gpiochip96
```

The gpiochipN folders provide information about single gpio controllers registered on the board:

```
~# ls /sys/class/gpio/gpiochip0
base      label      ngpio      subsystem  uevent
```

The entry base contains information about the base GPIO number and ngpio contains the whole amount of GPIOs provided by the GPIO controller.

We use GPIO19 as an example to show the usage of single GPIOs.

```
~# echo 19 > /sys/class/gpio/export
```

This way we export gpio19 fo userspace usage. If the export was succesful, we will find a directory named /sys/class/gpio/gpio19 afterwards. Within this directory we will be able to find the entries to access the functions of this gpio.

If we wish to set the direction and initial level of the GPIO, we can use the command:

```
~# echo "high" > /sys/class/gpio19/direction
```

This way we export GPIO19 for userspace usage and define our GPIO's direction attribute to an output with initially high level. You can change the value or direction on this GPIO by using the entries direction or value. Note that this method is not very fast, so for quickly changing GPIOs it is still necessary to write a driver. The method works fine for example to influence an LED directly from userspace.

To unexport an exported GPIO write the corresponding pinnumber into the /sys/class/gpio/export.

```
~# echo 19 > /sys/class/gpio/unexport
```


4.5 GPIO Events

Some GPIOs are able to issue an interrupt. The interrupts generated by the gpios can be registered to the UIO (Userspace IO) Subsystem. As default no GPIO is registered. The UIO Subsystem creates a device node in devfs for every registered uio device. To enable interrupt collecting write a non-zero value into `/dev/uio0`. Reading the device `/dev/uio0` returns an int value which is the event count (number of interrupts) seen by the device or a read error if no changes have occurred since last read. If you open the devices in a blocking way, the read operation will block until an interrupt happens. The file descriptor can be passed to `poll()`.

If you want to register events from GPIOs to the uio subsystem. You can edit the board file in kernel. Look at the file `arch/arm/mach-pxa/pcm969_990-baseboard.c` in the kernel tree and refer to the struct `pcm990_gpioevent_uio_info`.

4.6 Network

The phyCORE-PXA270 module has an SMSC 91C111 ethernet chip onboard, which is being used to provide the etho network interface. The interface offers a standard Linux network port which can be programmed using the BSD socket interface.

4.7 LCD Graphics

phyCORE-PXA270's LCD support uses the standard PXA2XX's framebuffer support and can be used as a regular console when also an USB keyboard is attached to the system. fb-tools can be used to manipulate the frame buffer (colour depth).

Normally a Hitachi tx09d70vm1cca is mounted on the PCM969 board. Additionally you can choose the LCD type with kernel command line during boot time. The kernel command line should be however already set up correctly in the u-boot environment if you have followed this Quickstart. So mostly you won't have to change this. For display definitions (resolution and frequency) see source file `arch/arm/mach-pxa/pcm969_990-baseboard.c` in the kernel tree.

4.8 LCD Backlight

The LCD backlight can be controlled by using the backlight class driver. This driver offers a sysfs entry to control the brightness and a connection to the frame buffer console and to the X-server for power management.

You can find the sysfs entries in `/sys/class/backlight/pwm-backlight` and control them with plain numbers.

- `max_brightness`
To read back the maximum value (hardware dependend). This value feeded into the brightness entry gives the maximum backlight brightness.
- `brightness`
Set the current brightness value (0 ...max_brightness).

- `bl_power`
Set or read back backlight power. 0 means backlight is off, 1 means on.
- `actual_brightness`
To read back the current brightness setting. Its the same as brightness.

4.9 SPI

The phyCORE-PXA270 board supports an SPI bus, based on the PXA270's integrated SPI controller. It is connected to the onboard devices using the standard kernel method, so all methods described here are not special to the phyCORE-PXA270.

On the phyCORE-PXA270, channel 1 of the SPI controller is connected to the MAX7301 GPIO expander chip. The BSP currently uses the "Chip Select" alternate function of GPIO 24 to select the MAX7301; This mean the controller handles chip selection by its own in hardware. This SPI controller mode works fine if only one SPI slave device is connected (in the case of phyCORE-PXA270 it is the MAX7301, see below).

If its planned in a custome design to add more devices to this SPI channel 1 (to let it act like a bus) any chip selection has to be done in software. In this case also for the MAX7301, so GPIO 24 must be a regular GPIO without any alternate function enabled.

For a description of the SPI framework see [Documentation/spi/spi-summary](#) and for PXA2xx's SPI driver see [Documentation/spi/pxa2xx](#)

4.10 GPIO Expander

This is a GPIO expander that supports 28 additional EGPIOs.

You can find the gernerel inforation of the gpio expander chip in `/sys/class/gpio/gpiochip228/`.

To control the direction and level of each EGPIO please use the gpiolib sysfs entries as described in section [4.4](#).

Note: The GPIO base number of MAX7301 expander is 228. So do remember to add 228 to the single gpio number on the expander when you want to control it. e.g. GPIO2 on the MAX7301 expander would be GPIO230 on the system.

If the modules for camera interface is loaded. There might be a further GPIO expander that supports 4 additional GPIOs.

You can find the general information of the GPIO expander chip in `/sys/class/gpio/gpiochip129/`.

4.11 AC97 Based Audio

The sound features can be used through standard PXA2xx AC97 ALSA support for the onboard Wolfson WM9712 device. See sources in `sound/arm/pxa2xx.c` in the kernel source tree for further information.

4.11.1 Sound Output

To play a sound, copy our favorite mp3 file to the phyCORE-PXA270, pop up the volume and play the file.

```
~# amixer sset PCM,0 20,20 unmute
~# amixer sset Headphone,0 20,20 unmute
~# amixer sset Master,0 20,20 unmute      # control the speaker
~# amixer sset 'Master Left Inv',0 on     # activate the speaker by phase reversal
~# madplay <mp3file_name>
```

If external loudspeakers are connected it is possible to mute the built-in speaker with `amixer sset 'Master Left Inv',0 off`.

Note: We also can use the command "alsamixer" to handle mixer's settings.

4.11.2 Sound Record

Note: When the Wolfson WM9712 chip comes up after power on, every sound source is muted as default. To record any sound the desired audio source must be unmuted first.

To activate sound capturing the internal ADCs have to be powered up and unmuted first:

```
~# amixer sset ADC,0 on
~# amixer sset Capture 15,15 unmute
```

Now its time to select the desired audio source for capturing. The following commands select the stereo line in as the source:

```
~# amixer sset Line 30,30 unmute
~# amixer sset 'Capture Select',0 Line
```

To select the microphone instead of the stereo line in, these commands are required:

```
~# amixer sset 'Mic 1',0 30
~# amixer sset Capture Select,0 'Mic 1'
```

Maybe the recorded sound level will be very low. To improve the volume we can enable a 20dB boost with the following command:

```
~# amixer sset 'Capture 20dB Boost',0 on
```

To record any sound the command `arecord` is the recommended way to do it. This example records about 20 seconds from the desired source:

```
~# arecord -f dat -d 20 -D hw:0,0 test.wav
```

See `arecord`'s manual for further meaning of the command line parameters.

4.11.3 Advanced Sound Handling

Note: The Wolfson WM9712 is a complex beast with many features. Sometimes it's hard to understand why it works or why it fails. Armed with its datasheet, the AC'97 specification and the kernel's powerful AC97 debug feature it is much easier to use WM9712 features in the manner we like or the way the chip supports it. Not all WM9712 features are supported by the ALSA utils out of the box. Some of these features need kernel driver patches to make the ALSA utils aware of it.

To see the current WM9712 register settings simply enter:

```
~# cat /proc/asound/card0/codec97#0/ac97#0-0+regs
```

This is an easy way to check the results of the `amixer` command and if it supports this feature out of the box.

To change any register's value manually (without `amixer` command for test purposes only) simply enter:

```
~# echo "1a 0404" > /proc/asound/card0/codec97#0/ac97#0-0+regs
```

This example updates WM9712's register `0x1A` to the new value `0x0404`. You will also need the datasheet here to know the registers, their offset and meaning.

Note: Give all values in hex but without leading `0x`.

4.12 AC97 Based Touchscreen

This device is supported through PXA2xx's standard AC97 support for the onboard Wolfson WM9712 device driver for touchscreen. In userspace this device is supported through the `tslib`, so it can be used by an X server as a pointing device. See sources in `driver/input/touchscreen/wm97xx.c` in the kernel source tree for further information.

Modul parameters to control the driver:

- **cont_rate** Sample rate in continuous mode (Hz).
Default is 200 samples per second.
- **pen_int** Pen down detection (1 = interrupt, 0 = polling).
This driver can either poll or use an interrupt to indicate a pen down event. If the IRQ request fails, then it will fall back to polling mode. Default is interrupt.
- **pressure** Pressure readback (1 = pressure, 0 = no pressure).
- **ac97_touch_slot** Touch screen data slot AC97 number.
enable/disable AUX ADC sysfs, default is enabled
- **aux_sys** disable AUX ADC sysfs entries.
- **status_sys** disable codec status sysfs entries.
enable/disable codec status sysfs, default is enabled
- These parameters are used to help the input layer discard out of range readings and reduce jitter etc.
 - min, max: indicate the min and max values our touch screen returns

- **fuzz**: use a higher number to reduce jitter

The default values correspond to Mainstone II in QVGA mode Please read Documentation/input/input-programming.txt for more details.

- **abs_x** Touchscreen absolute X min, max, fuzz.
- **abs_y** Touchscreen absolute Y min, max, fuzz.
- **abs_p** Touchscreen absolute Pressure min, max, fuzz.
- **rpu** Set internal pull up resistor for pen detect.
Pull up is in the range 1.02k (least sensitive) to 64k (most sensitive) i.e. pull up resistance = 64k Ohms / rpu.
We adjust this value if we are having problems with pen detect not detecting any down event.
- **pil** Set current used for pressure measurement.
Set
 - pil = 2 to use 400µA
 - pil = 1 to use 200µA and
 - pil = 0 to disable pressure measurement.

This is used to increase the range of values returned by the ADC when measuring touchpanel pressure.

- **pressure** Set threshold for pressure measurement.
Pen down pressure below threshold is ignored.
- **delay** Set ADC sample delay.
For accurate touchpanel measurements, some settling time may be required between the switch matrix applying a voltage across the touchpanel plate and the ADC sampling the signal.
This delay can be set by setting delay = n. Valid values of n can be looked up in the 'delay_table' in the driver source. Long delays >1ms are supported for completeness, but are not recommended.
- **five_wire** Set to '1' to use 5-wire touchscreen.
NOTE: Five wire mode does not allow for readback of pressure.
- **mask** Set ADC mask function.
Sources of glitch noise, such as signals driving an LCD display, may feed through to the touch screen plates and affect measurement accuracy. In order to minimise this, a signal may be applied to the MASK pin to delay or synchronise the sampling.
 - 0 = No delay or sync
 - 1 = High on pin stops conversions
 - 2 = Edge triggered, edge on pin delays conversion by delay param (above)
 - 3 = Edge triggered, edge on pin starts conversion after delay param

Using the touchscreen requires a calibration. This has to be done the first time a newly built OSELAS.BSP-Phytec-phyCORE-9 runs on the target to create the calibration information before we can use the X server.

To do so run the command:

```
~# ts_calibrate
```

The command uses the environment variable TSLIB_TSDEVICE (defined in /etc/profile) and the so called ts-lib, configured in /etc/ts.conf.

Note: When we intend to calibrate the touchpanel, stop an already running X server prior to starting `ts_calibrate`. They can't share the framebuffer, so the X server gets killed and the `ts_calibrate` command might hang forever.

4.13 USB Host Controller

Standard OHCI Rev. 1.0a implementation.

Only channel 1 is supported, channel 2 and 3 are not available.

Make sure the required USB device module for the device to be attached is already loaded. The OSELAS.BSP-Phytec-phyCORE-g supports USB mice and USB Mass Storage devices (MemorySticks aso.) as default.

4.14 USB Gadget

Linux can run USB protocol in the host, or in the peripheral mode. With OSELAS.BSP-Phytec-phyCORE-g BSP the USB peripheral mode is supported as well. This means that we can connect our phyCORE-PXA270 to a USB host. Depending on which driver we load the phyCORE-PXA270 can act as various USB gadget. At default the file backed storage gadget driver and the network gadget driver are shipped. In the following is a short description about how to use the file backed storage gadget. For more information about USB gadgets please consult <http://www.linux-usb.org/gadget/>:

The File-backed Storage Gadget (FSG) is provided by the module `g_file_storage`. It can appear to a host as a set of SCSI disk drives. The information stored for each LUN must be maintained by the gadget somewhere, either in a normal file or in a block device. This file or block device is called the "backing storage" for the gadget and must be given as module parameter while loading the `g_file_storage` driver. You can e.g export our system partition with

```
~# modprobe g_file_storage file=/dev/mtdblock2
```

A "backing storage" can also be created as a normal file. Please follow the steps below to create a suitable backing storage file.

```
~# dd bs=1M count=8 if=/dev/zero of=/home/backingstore
8+0 records in
8+0 records out
~# fdisk -S 8 -H 16 -C 32 /home/backingstore
```

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-128, default 1):
Using default value 1
```

Last cylinder, +cylinders or +size{K,M,G} (1-32, default 32):
Using default value 32

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): b
Changed system type of partition 1 to b (W95 FAT32)

Command (m for help): w
The partition table has been altered!

This way we have created a 8 MiB big backing store file containing a single partition marked as vfat. Feel free to change the file size. Just change the corresponding parameter in `dd` and `fdisk` to do so.

As the next step we bind the file to a loop interface, make a vfat file system on it, mount it, write a test file and unmount it again.

```
~# modprobe loop
~# mkdosfs /dev/loop0
~# losetup -o 4096 /dev/loop0 /home/backingstore
~# mkdosfs /dev/loop0
mkdosfs 3.0.1 (23 Nov 2008)
Loop device does not match a floppy size, using default hd params
~# mount -t vfat /dev/loop0 /mnt
~# touch /mnt/testfile
~# umount /mnt
~# losetup -d /dev/loop0
```

After running the steps above we can finally load the `g_file_storage` driver.

```
~# modprobe g_file_storage file=/home/backingstore
```

Now connect the USB Micro B interface on our board with a host side USB interface. The host should now detect a `usb-mass-storage` device. You can mount it like a common USB stick.

Below an example of the host-side output is shown:

```
user@host ~# dmesg | tail -17
usb 5-2: new full speed USB device using uhci_hcd and address 36
usb 5-2: configuration #1 chosen from 1 choice
scsi26 : SCSI emulation for USB Mass Storage devices
usb-storage: device found at 36
usb-storage: waiting for device to settle before scanning
usb-storage: device scan complete
scsi 26:0:0:0: Direct-Access      Linux      File-Stor Gadget 0311 PQ: 0 ANSI: 2
sd 26:0:0:0: [sd] 16384 512-byte hardware sectors (8 MB)
sd 26:0:0:0: [sd] Write Protect is off
sd 26:0:0:0: [sd] Mode Sense: 0f 00 00 00
```

```
sd 26:0:0:0: [sd] Assuming drive cache: write through
sd 26:0:0:0: [sd] 16384 512-byte hardware sectors (8 MB)
sd 26:0:0:0: [sd] Write Protect is off
sd 26:0:0:0: [sd] Mode Sense: 0f 00 00 00
sd 26:0:0:0: [sd] Assuming drive cache: write through
   sdc: sdc1
sd 26:0:0:0: [sd] Attached SCSI disk
user@host ~# sudo mount /dev/sdc1 /mnt
user@host ~# ls /mnt
testfile
```



The USB Gadget support on PXA270 Platforms is still very experimental. Do expect unusual behaviours of the system while using this. Because of this we don't recommend the usage of the USB Gadget system in production field.

4.15 I²C Master

The PXA270 processor based phyCORE-PXA270 supports a dedicated I²C controller on chip. The kernel supports this controller as a master controller.

Additional I²C device drivers can use the standard I²C device API to gain access to their devices through this master controller.

For further information about the I²C framework see `Documentation/i2c` in the kernel source tree.

4.15.1 I²C Realtime Clock (RTC8564)

Due to the kernel's Real Time Clock framework the RTC8564 clock chip can be accessed using the same tools as any other clocks.

Date and time can be manipulated with the `hwclock` tool, using the `-systohc` and `-hctosys` options. For more information about this tool refer to `hwclock`'s manpages.

4.15.2 I²C device 24W32

This device is a 4kByte non-volatile memory. Only the upper 2kByte can be used for any purpose, due to the lower 2kBytes being used by the bootloader itself to store its environment.

This type of memory is accessible through the `sysfs` filesystem. To read the EEPROM contents simply `open()` the entry `/sys/bus/i2c/devices/0-0054/eeprom` and use `fseek()` and `read()` to get the values.

4.16 Matrix keypad support

The phyCORE-PXA270 is shipped with an interface for a matrix keypad, which can hold 10 keys. Additionally there are three keys premounted on the board. At default the three onboard keys are defined as UP/DOWN/ENTER. The keys of the additional keypads will be used as 0-9 number keys. The keypad interface is supported by the driver `pxa27x_keypad`. You can find the device file for this interface at `/dev/input/event1`. To test the interface, just do a

```
~# cat /dev/input/event1
```

and press some keys. If the interface works properly, you will see some random characters on your command prompt.

4.17 Status LEDs

There are 6 user programmable LEDs on phyCORE-PXA270. Four of them are connected to GPIOs and two are driven by PWM. For all six LEDs the LED class driver is used. You can find the sysfs entries in `/sys/class/leds/`

```
~# ls /sys/class/leds/
pcm969:green:pwm1      pcm969:white:pwm0      pcm969:yellow:user3
pcm969:red:user1       pcm969:yellow:user2    pcm969:yellow:user4
```

The name of the folders indicates the color and name of the LEDs. The LEDs with the namescheme `"pcm969:[yellow|red]:user[0|1|2|3]"` are GPIO connected LEDs. The other two are driven by PWM. The folders contain entries you can use to control the single LEDs.

```
~# ls /sys/class/leds/pcm969:red:user1
brightness device      subsystem uevent
```

The brightness file will set the brightness of the LED (taking a value 0-255). The GPIO connected LEDs don't have brightness support so will just be turned on for non-zero brightness settings.

4.18 Camera Interface

In this section we will handle the usage of the camera interface. Depending on what we want to use the camera for we have numerous possibilities here combining different hardware as well as software configuration.

4.19 Using the Camera Support

Before we start with details, we should know that along with our OSELAS.BSP-Phytec-phyCORE-9 BSP comes a set of preconfigured scripts which are capable to do some basic guessing what kind of hard/software we have and start the camera capturing with proper parameters.

Nevertheless one should go through the following sections to understand how the stuff exactly works.

We can find the convenience scripts on our target under `/home/phycore-gst-examples-1.1.0`. The version number in the path may vary.

```
~# ls /home/phycore-gst-examples-1.1.0/
bwcam-fbdev_240x320  bwcam-fbdev_640x480  bwcam-xv
colcam-fbdev_240x320  colcam-fbdev_640x480  colcam-xv func.sh
```

The name of the scripts stands for for what they can be used. If we e.g have a b/w Camera and a 240x320 display on our board, we can start capturing by simply doing:

```
~# cd /home/phycore-gst-examples-1.1.0/
~# ./bwcam-fbdev_240x320
removing old drivers ...
loading bw cam drivers ...
camera 0-0: PXA Camera driver attached to camera 0
camera: probe of 0-0 failed with error -121
camera 0-0: PXA Camera driver attached to camera 0
camera 0-0: Detected a MT9V022 chip ID 1313, monochrome sensor
camera 0-0: PXA Camera driver detached from camera 0
starting gstreamer ...
camera 0-0: PXA Camera driver attached to camera 0
New clock: GstSystemClock
```

The occuring error messages can be ignored: The script tries to probe for a suitable camera module. We will only see them if we access the board through a serial terminal. If everything goes well, there won't be any further messages on the console and a camera picture should be visible on the display. We can stop capturing and displaying by pressing the well known `ctrl+c`.

Below a list of scripts the OSELAS.BSP-Phytec-phyCORE-9 BSP provides and their intentional usage:

- `bwcam-fbdev_640x480` for b/w camera+640x480 display
- `colcam-fbdev_240x320` for colour camera + 240x320 display
- `colcam-fbdev_640x480` for colour camera + 640x480 display
- `bwcam-xv` for b/w camera + X enabled display
- `colcam-xv` for colour camera + X enalbed display

4.20 Turn on Camera Support In System

The phyCORE-PXA270 comes with a camera, which allows us to use the video capture interface. Phytec ships different kind of cameras with their development kits. Depending on the kit we might need different driver for these cameras. There are two camera drivers we can choose:

mt9v022 This driver supports *Micron MT9Vo22* picture sensors. If we have a camera of the VM-007 series, we will most probably need this driver.

mt9m001 This driver supports *Micron MT9M001* picture sensors, as built in cameras of the VM-006 series.



The `mt9v022` driver for the *Micron MT9V022* picture sensor is not capable to detect if the camera comes with a colour or monochrome sensor. It has to be defined with module parameter `"sensor_type"` manually

When loaded the driver tries to detect a camera, which it supports. If the check is successful, the driver will register the camera device to the video subsystem and creates a device node name `/dev/videoX` in the file system, where the `X` stands for the numbering of the device. If we have one camera only, which is mostly the case, it should be `/dev/video0`. If our system misses this file, we have most probably loaded the wrong driver or our camera is not attached properly.

As default the driver `mt9v022` is loaded with b/w support. If we have another camera attached to our kit, we have to load a different camera driver or reload our driver with another module parameter. To do so, we must unload the current driver first:

```
~# rmmod mt9v022
```

Note: Before we proceed we have to ensure the module `pca953x` is loaded, which must be present before any camera module can be loaded. We can run the following command to check it:

```
~# lsmod | grep -o pca953x
pca953x
```

If we don't get any output, we must load the `pca953x` module manually first:

```
~# modprobe pca953x
```

If we want to load `mt9v022` with support for colour sensors, we can reload the same module with the following module parameter.

```
~# modprobe mt9v022 sensor_type=colour
```

In contrast to the `mt9v022` kernel module the `mt9m001` kernel module doesn't have any module parameters. If we want to load it, we just do:

```
~# modprobe mt9m001
```

4.21 Accessing Camera With GStreamer

Once we have initialized the camera support in our system properly, we can use *GStreamer* to access the video stream. *GStreamer* is a streaming media framework, based on graphs of filters which operate on media data. *GStreamer* consists of several command line tools and sets of plugins. Generally there are three kind of plugins:

Source Plugins These plugins access directly media sources on the system and pass the media data on to further plugins. For example, we can use the `v4l2src` plugin to access the camera.

Pipe Plugins These plugins are like double ended tubes. We can put data in one side and take the processed data from the other side.

Sink Plugins These plugins take the media data and put them to "sinks", which are output devices like frame-buffer device, x.org client or even Network sockets.

4.22 Basic Usage Of GStreamer

Using *GStreamer* on command line is like building a pipeline. We need a source plugin at the beginning, some pipe plugins in the middle and a sink plugin at the other end. To launch the pipeline, we need the command-line tool `gst-launch`.

4.22.1 Simple Usage Example

To get a feeling how this works we can use the built-in fake plugins to build a pipeline, which simply process some empty buffers:

```
~# gst-launch -v fakesrc num-buffers=1 ! fakesink
```

This will print out output that looks similar to this:

```
Setting pipeline to PAUSED ...
/pipeline0/fakesrc0: last-message = "get      ***** > (    0 bytes, timestamp: none,
duration: none, offset: 0, offset_end: -1, flags: 0) 0x42e68"
/pipeline0/fakesink0: last-message = "preroll ***** "
/pipeline0/fakesink0: last-message = "event ***** E (type: 102, GstEventNewsegment,
update=(boolean>false, rate=(double)1, applied_rate=(double)1,
format=(GstFormat)GST_FORMAT_BYTES, start=(gint64)0, stop=(gint64)-1,
position=(gint64)0;) 0x3ccc0"
Pipeline is PREROLLING ...
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
/pipeline0/fakesink0: last-message = "chain ***** < (    0 bytes,
timestamp: 0:00:00.000000000, duration: none, offset: 0,
offset_end: -1, flags: 32) 0x42e68"
/pipeline0/fakesink0: last-message = "event ***** E (type: 86, ) 0x3ccc0"
New clock: GstSystemClock
Got EOS from element "pipeline0".
Execution ended after 4653538 ns.
Setting pipeline to PAUSED ...
Setting pipeline to READY ...
Setting pipeline to NULL ...
FREEING pipeline ...
```

All plugins are connected with an exclamation mark (!)

If this works properly, we can go on to do some real work in the next section.

4.22.2 Simple Monochrome Usage Example

The following line will grab the video stream from a monochrome camera and put it on the framebuffer device:

```
~# gst-launch v4l2src ! video/x-raw-gray ! ffmpegcolorspace ! fbdevsink
```

Three plugins are used here:

1. v4l2src plugin grabs the raw frames from the camera using the Video4Linux2 API
2. video/x-raw-gray in the pipeline is a capability filter. It sets a mime type to specify a desired video format (in this case grayscale)



Some cameras do not provide their framesize properly to the system or just simply provide a frame bigger than the system can process. In this case the command will fail, and we have to define the framesize manually. Refer section 4.23.2 on page 38 to find out how to do it.

3. ffmpegcolorspace takes the stream and converts it to suitable colourspace
4. fbdevsink takes the converted stream and displays it on a framebuffer device

4.22.3 Simple Colour Usage Example

If we have a colour sensor instead, we can use the following line to process the videostream:

```
~# gst-launch v4l2src ! video/x-raw-bayer ! bayer2rgb bg_first=0 \ 
> ! ffmpegcolorspace ! fbdevsink
```

Here we are using the bayer2rgb plugin to convert the raw content provided by the camera sensor into an RGB signal.



The bayer2rgb plugin needs a parameter to define if the first line of the bayer pattern provided by the sensor is a blue/green line or a green/red line. We can try to change this to 1 if we experience any troubles with our colourspace.

4.23 Advanced Usage Of GStreamer

Here are the details:

4.23.1 Manual Setting for Frame Rate and Framesize

Along with the mimetype definition we can also optionally set information like frame size and rate. e.g.

```
~# gst-launch v4l2src \   
> ! video/x-raw-gray,width=640,height=480,framerate=30/1 \   
> ! ffmpegcolospace ! fbdevsink
```

Doing this will set a fix value for the input frame format. Such options may come handy if we e.g. have trouble with the size of the input stream. Note:

- If we adjust the width and height ourself, the shown region starts at the upperlefter corner of the frame captured by the camera.
- The frame size we can choose depends on the one our camera can provide. If we choose any size which extends the picture range of the camera, *GStreamer* will fail to start with an output like this:

```
ERROR: from element /pipeline0/v4l2src0: Could not negotiate format
```

4.23.2 Manipulate Input Frame Size with Plugins

Normally the onboard framebuffer device has smaller size than the picture captured by the camera. Because of that, only a part of the captured video might be visible, starting at the upleft corner, on the display. To get the picture we actually want on our display, we can use various plugins:

- we can crop the videosignal using the `videocrop` plugin

```
~# gst-launch v4l2src ! video/x-raw-gray \   
> ! videocrop left=250 right=250 top=80 bottom=80 \   
> ! ffmpegcolospace ! fbdevsink
```

Will "chop out" 500 pixel in the horizontal and 160 pixel in the vertical direction of the input frame, so that we will get a 250x320 sized frame which is positioned in the central of the input frame. This command line will work with an MT9v022 camera, which provides a 752(H)x480(V) sized frame at default and a Hitachi tx0gd7ovm1cca, which is 240(H)x240(V) in size. We should consult the datasheet of our display and our camera to find out the correct crop parameter for the camera kit.

- If we'd rather prefer not to crop out regions of the input frame, we can use the `videoscale` plugin to resize the input frame

```
~# gst-launch v4l2src ! video/x-raw-gray \   
> ! ffmpegcolospace ! videoscale ! video/x-raw-yuv,width=320,height=240 ! \   
> ! ffmpegcolospace ! fbdevsink
```

With this command we can resize the video frame to 320x240. Notice:

- Unlike the `videocrop` plugin, `videoscale` cannot process raw data provided by `v4l2src` directly. So we put a `ffmpegcolospace` between the `v4l2src` and `videoscale`.
- Aspect ration in the resized frame should be proportional to the original size. Otherwise we might get a disorted image.

4.23.3 Manipulate Picture's Orientation

Further we can flip and rotate our video with the `videoflip` plugin. We take the command above as example:

```
~# gst-launch v4l2src ! video/x-raw-gray \   
> ! ffmpegcolorspace \   
> ! videoscale ! video/x-raw-yuv,width=320,height=240 \   
> ! videoflip method=clockwise \   
> ! ffmpegcolorspace ! fbdevsink
```

The additional `videoflip` plugin in the pipeline flips the video 90 degrees clockwise.

4.23.4 Using other Sinks than the Framebuffer

Beside the local framebuffer there are additional locations where to show the video stream.

4.23.4.1 ximagesink and xvimagesink

If the support for X protocols is turned on in our OSELAS.BSP-Phytec-phyCORE-9, we can use it to display our camera stream on a remote host, which runs a common X server like X.Org. To do this we run the following steps:

On our host:

```
user@host ~ xhost +  
access control disabled, clients can connect from any host
```



Note: With this command we will grant access from any X-Clients in our network to our local X-Server. This could be a security issue. Hence we might probably want to consult our system administrator first before doing this.

On target:

```
~# export DISPLAY=[IP of our host in here]:0
```

Now any X-related application started on our target will be shown on our host and we can start our *GStreamer* with the `ximagesink`

```
#~ gst-launch v4l2src ! video/x-raw-gray ! ffmpegcolorspace ! ximagesink
```

Notice that displaying our videostream like this we might get very low framerate due to high network load.

4.23.4.2 udpsink

With *GStreamer* we can also stream our video with various streaming protocols through the network. However we have to encode the stream with a video codec first. Currently only hardware encoding is supported. We only can use this feature if our phyCORE-PXA270 provides a video processing unit.

If our target system supports such a feature, we can start streaming on the target with following command:

```
~# gst-launch v4l2src \   
> ! video/x-raw-gray ! ffmpegcolospace \   
> ! mfw_vpuencoder \codec-type=std_avc bitrate=32767 gopsize=10 ! rtph264pay \   
> ! udpsink host=[HOST IP ADDRESS] port=5555
```

On the host we can use the following command to decode this video stream:

```
user@host ~ gst-launch udpsrc port=5555 \   
> caps = "application/x-rtp, media=(string)video" ! rtph264depay \   
> ! ffdec_h264 ! xvimagesink
```


5 Getting help

Below is a list of locations where you can get help in case of trouble. For questions how to do something special with PTXdist or general questions about Linux in the embedded world, try these.

5.1 Mailing Lists

About PTXdist in particular

This is an English language public mailing list for questions about PTXdist. See

http://www.pengutronix.de/maillinglists/index_en.html

how to subscribe to this list. If you want to search through the mailing list archive, visit

<http://www.mail-archive.com/>

and search for the list *ptxdist*.

About embedded Linux in general

This is a German language public mailing list for general questions about Linux in embedded environments. See

http://www.pengutronix.de/maillinglists/index_de.html

how to subscribe to this list. Note: You also can send mails in English.

5.2 News Groups

About Linux in embedded environments

This is an English newsgroup for general questions about Linux in embedded environments.

comp.os.linux.embedded

About general Unix/Linux questions

This is a German newsgroup for general questions about Unix/Linux programming.

de.comp.os.unix.programming

5.3 Chat/IRC

About PTXdist in particular

irc.freenode.net:6667

Create a connection to the **irc.freenode.net:6667** server and enter the chatroom **#ptxdist**. This is an English room to answer questions about PTXdist. Best time to meet somebody there is at European daytime.

5.4 Miscellaneous

Online Linux Kernel Cross Reference

A powerful online cross reference.

<http://lxr.linux.no/>

U-Boot manual (partially)

Manual how to survive in an embedded environment and how to use the U-Boot on target's side

<http://www.denx.de/wiki/DULG>

5.5 phyCORE-PXA270 Support Maillist

OSELAS.Phytec@pengutronix.de

This is an english language public maillist for all BSP related questions specific to Phytex's hardware. See web site

http://www.pengutronix.de/maillinglists/index_en.html

5.6 Commercial Support

You can order immediate support through customer specific mailing lists, by telephone or also on site. Ask our sales representative for a price quotation for your special requirements.

Contact us at:

Pengutronix
Peiner Str. 6-8
31137 Hildesheim
Germany
Phone: +49 - 51 21 / 20 69 17 - 0
Fax: +49 - 51 21 / 20 69 17 - 55 55

or by electronic mail:

sales@pengutronix.de